

EAST Search History

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	2	"20040268096".pn.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/28 10:29
L2	2	"20040268095".pn.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/28 10:32
L3	807	707/206.ccls.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/28 11:13
L4	27	l3 and (read adj barrier)	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/28 11:24
L5	11	l4 and null	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/28 11:26
L6	787	717/140.ccls.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/28 11:27
L7	608	717/151.ccls.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/28 11:27
L8	2	l6 and (read adj barrier)	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/28 11:37

EAST Search History

L9	1	L7 and (read adj barrier)	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/28 11:28
L10	108	("5857210" "5873105" "5873104" "4278830" "4321099" "5022269" "5372451" "5438539" "6049810" "4313462" "4318050" "4340079" "4377755" "4393479" "4408697" "4442984" "4478391" "4517779" "4548095" "4566242" "4756134" "4763176" "4797548" "4878690" "4931845" "4961194" "4971071" "4987562" "5009308" "5182788" "5201327" "5256579" "5293614" "5302225" "5395774" "5432356" "5536667" "5563581" "5606176" "5616942" "5738565" "5765319" "5807198" "5809733" "5833897" "5854114" "5871842" "5878836" "5891775" "5898350").pn.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/28 11:37
S1	2	"20050149588".pn.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/07 12:17
S2	0	read adj barrier adj handler	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/13 13:32
S3	72	read adj barrier	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/07 12:18
S4	46	S3 and @ad < "20040331"	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/13 13:34

EAST Search History

S5	3	S4 and recovery	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/07 12:43
S6	369	speculative adj load	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/07 12:43
S7	2	S3 and S6	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/07 12:45
S8	2	"6880073".pn.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/07 12:46
S9	1	read with barrier with fault with handl\$3	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/13 13:33
S10	29	(read with barrier) and (fault with handl\$3)	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/13 13:35
S11	25	S10 and @ad < "20040331"	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/13 13:35
S12	7	(read adj barrier) and (fault with handl\$3)	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/13 13:35

EAST Search History

S13	6	S12 and @ad < "20040331"	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/27 15:49
S14	46	(read adj barrier) and (bit)	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/27 15:49
S15	30	S14 and @ad < "20040331"	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/11/28 10:29

[Home](#) | [Login](#) | [Logout](#) | [Access Information](#) | [Alerts](#) | [Purchase History](#) | [Cart](#)

Welcome United States Patent and Trademark Office

☐ Search Session History[BROWSE](#)[SEARCH](#)[IEEE XPLORE GUIDE](#)

Wed, 28 Nov 2007, 10:56:29 AM EST

Edit an existing query or
compose a new query in the
Search Query Display.

Search Query Display

Select a search number (#)
to:

- Add a query to the Search Query Display
- Combine search queries using AND, OR, or NOT
- Delete a search
- Run a search



Recent Search Queries

- #1 ((read barrier)<in>metadata)
- #2 ((read barrier)<in>metadata)
- #3 ((read <near> barrier)<in>metadata)
- #4 ((read <near> barrier)<in>metadata)
- #5 (((read <near> barrier) and (null <near> (reference <or> check)))<in>metadata)
- #6 (((read <near> barrier) and (null <near> (reference <or> check)))<in>metadata)
- #7 (((read <near> barrier) and (null))<in>metadata)



Indexed by
 Inspec[®]

[Help](#) [Contact Us](#) [Privacy &](#)

© Copyright 2007 IEEE -


[Subscribe \(Full Service\)](#) [Register \(Limited Service, Free\)](#) [Login](#)

 Search: ☒ The ACM Digital Library ☐ The Guide



THE ACM DIGITAL LIBRARY

[Feedback](#) [Report a problem](#) [Satisfaction survey](#)

 Terms used: read barrier

 Found **53,537** of **215,481**

Sort results by

☒ [Save results to a Binder](#)
[Try an Advanced Search](#)

Display results

☒ [Search Tips](#)
[Try this search in The ACM Guide](#)
☐ [Open results in a new window](#)

Results 1 - 20 of 200

 Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

Best 200 shown

 Relevance scale ☐ ☐ ☐ ☐ ☐

1 [Garbage collection: A true hardware read barrier](#)

Matthias Meyer

 June 2006 **Proceedings of the 5th international symposium on Memory management ISMM '06**

Publisher: ACM

 Full text available: [pdf\(991.66 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Read barriers synchronize compacting garbage collection and application processing in a simple yet elegant way. Unfortunately, read barrier checks are expensive to implement in software, and even with hardware support, the clustering of read barrier faults irregularly impairs application progress to an unacceptable extent. For this reason, read barriers are often considered unsuitable for hard real-time systems. In this paper, we introduce a novel hardware read barrier design for an object-based ...

Keywords: hardware support, object-based processor architecture, read barrier, real-time garbage collection

2 [The case for a read barrier](#)



Douglas Johnson

 April 1991 **ACM SIGPLAN Notices , ACM SIGARCH Computer Architecture News , ACM SIGOPS Operating Systems Review , Proceedings of the fourth international conference on Architectural support for programming languages and operating systems ASPLOS-IV, Volume 26 , 19 , 25 Issue 4 , 2 , Special Issue**

Publisher: ACM Press

 Full text available: [pdf\(881.24 KB\)](#) Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)

3 [Implementation techniques: Barriers: friend or foe?](#)



Stephen M. Blackburn, Antony L. Hosking

 October 2004 **Proceedings of the 4th international symposium on Memory management ISMM '04**

Publisher: ACM Press

 Full text available: [pdf\(137.10 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Modern garbage collectors rely on read and write barriers imposed on heap accesses by

the mutator, to keep track of references between different regions of the garbage collected heap, and to synchronize actions of the mutator with those of the collector. It has been a long-standing untested assumption that barriers impose significant overhead to garbage-collected applications. As a result, researchers have devoted effort to development of optimization approaches for elimination of unnecessary ...

Keywords: garbage collection, java, memory management, write barriers

4 Barrier techniques for incremental tracing



Pekka P. Pirinen

October 1998 **ACM SIGPLAN Notices , Proceedings of the 1st international symposium on Memory management ISMM '98**, Volume 34 Issue 3

Publisher: ACM Press

Full text available: [pdf\(707.56 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

This paper presents a classification of barrier techniques for interleaving tracing with mutator operation during an incremental garbage collection. The two useful tricolour invariants are derived from more elementary considerations of graph traversal. Barrier techniques for maintaining these invariants are classified according to the action taken at the barrier (such as scanning an object or changing its colour), and it is shown that the algorithms described in the literature cover all the poss ...

5 Implementation techniques: Exploring the barrier to entry: incremental generational garbage collection for Haskell



A. M. Cheadle, A. J. Field, S. Marlow, S. L. Peyton Jones, R. L. While

October 2004 **Proceedings of the 4th international symposium on Memory management ISMM '04**

Publisher: ACM Press

Full text available: [pdf\(458.55 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

We document the design and implementation of a "production" incremental garbage collector for GHC 6.2. It builds on our earlier work (Non-stop Haskell) that exploited GHC's dynamic dispatch mechanism to hijack object code pointers so that objects in to-space automatically scavenge themselves when the mutator attempts to "enter" them. This paper details various optimisations based on code specialisation that remove the dynamic space, and associated time, overheads that accompanied our earlier sch ...

Keywords: incremental garbage collection, non-stop haskell

6 Efficient techniques for fast nested barrier synchronization



Vara Ramakrishnan, Isaac D. Scherson, Raghu Subramanian

July 1995 **Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures SPAA '95**

Publisher: ACM Press

Full text available: [pdf\(806.60 KB\)](#) Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)


7 Concurrency: Write barrier elision for concurrent garbage collectors



Martin T. Vechev, David F. Bacon

October 2004 **Proceedings of the 4th international symposium on Memory management ISMM '04**

Publisher: ACM Press

Full text available:  [pdf\(490.73 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Concurrent garbage collectors require write barriers to preserve consistency, but these barriers impose significant direct and indirect costs. While there has been a lot of work on optimizing write barriers, we present the first study of their elision in a concurrent collector. We show conditions under which write barriers are redundant, and describe how these conditions can be applied to both incremental update or snapshot-at-the-beginning barriers. We then evaluate the potential for write b ...

Keywords: concurrent garbage collection, write barrier

8 Enforcing isolation and ordering in STM



Tatiana Shpeisman, Vijay Menon, Ali-Reza Adl-Tabatabai, Steven Balensiefer, Dan Grossman, Richard L. Hudson, Katherine F. Moore, Bratin Saha

June 2007 **ACM SIGPLAN Notices , Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation PLDI '07**, Volume 42 Issue 6

Publisher: ACM Press

Full text available:  [pdf\(257.39 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Transactional memory provides a new concurrency control mechanism that avoids many of the pitfalls of lock-based synchronization. High-performance software transactional memory (STM) implementations thus far provide *weak atomicity*: Accessing shared data both inside and outside a transaction can result in unexpected, implementation-dependent behavior. To guarantee isolation and consistent ordering in such a system, programmers are expected to enclose all shared-memory accesses inside tr ...

Keywords: code generation, compiler optimizations, escape analysis, isolation, ordering, strong atomicity, transactional memory, virtual machines, weak atomicity

9 An effective hybrid transactional memory system with strong isolation guarantees



Chi Cao Minh, Martin Trautmann, JaeWoong Chung, Austen McDonald, Nathan Bronson, Jared Casper, Christos Kozyrakis, Kunle Olukotun

June 2007 **ACM SIGARCH Computer Architecture News , Proceedings of the 34th annual international symposium on Computer architecture ISCA '07**, Volume 35 Issue 2

Publisher: ACM Press

Full text available:  [pdf\(239.24 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

We propose signature-accelerated transactional memory (SigTM), a hybrid TM system that reduces the overhead of software transactions. SigTM uses hardware signatures to track the read-set and write-set for pending transactions and perform conflict detection between concurrent threads. All other transactional functionality, including data versioning, is implemented in software. Unlike previously proposed hybrid TM systems, SigTM requires no modifications to the hardware caches, which reduces hardw ...

Keywords: multi-core architectures, parallel programming, strong isolation, transactional memory

10 An on-the-fly reference-counting garbage collector for java



Yossi Levroni, Erez Petrank

January 2006 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 28 Issue 1


Publisher: ACM Press

Full text available:  [pdf\(787.15 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Reference-counting is traditionally considered unsuitable for multiprocessor systems. According to conventional wisdom, the update of reference slots and reference-counts requires atomic or synchronized operations. In this work we demonstrate this is not the case by presenting a novel reference-counting algorithm suitable for a multiprocessor system that does not require any synchronized operation in its write barrier (not even a compare-and-swap type of synchronization). A second novelty of thi ...


Keywords: Programming languages, garbage collection, memory management, reference-counting.

11 [Sapphire: copying GC without stopping the world](#)

 Richard L. Hudson, J. Eliot B. Moss

June 2001 **Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande JGI '01**

Publisher: ACM Press

Full text available:  [pdf\(899.45 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)


Many concurrent garbage collection (GC) algorithms have been devised, but few have been implemented and evaluated, particularly for the Java programming language. Sapphire is an algorithm we have devised for concurrent copying GC. Sapphire stresses minimizing the amount of time any given application thread may need to block to support the collector. In particular, Sapphire is intended to work well in the presence of a large number of application threads, on small- to medium-scale shared memor ...

12 [Objects and their collection: The pauseless GC algorithm](#)

 Cliff Click, Gil Tene, Michael Wolf

June 2005 **Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments VEE '05**

Publisher: ACM Press

Full text available:  [pdf\(440.91 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Modern transactional response-time sensitive applications have run into practical limits on the size of garbage collected heaps. The heap can only grow until GC pauses exceed the response-time limits. Sustainable, scalable concurrent collection has become a feature worth paying for. Azul Systems has built a custom system (CPU, chip, board, and OS) specifically to run garbage collected virtual machines. The custom CPU includes a read barrier instruction. The read barrier enables a highly concurren ...


Keywords: Java, concurrent GC, custom hardware, garbage collection, memory management, read barriers

13 [A real-time garbage collector with low overhead and consistent utilization](#)

 David F. Bacon, Perry Cheng, V. T. Rajan

January 2003 **ACM SIGPLAN Notices , Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages POPL '03**, Volume 38 Issue 1

Publisher: ACM Press

Full text available:  [pdf\(517.37 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Now that the use of garbage collection in languages like Java is becoming widely accepted due to the safety and software engineering benefits it provides, there is significant

interest in applying garbage collection to hard real-time systems. Past approaches have generally suffered from one of two major flaws: either they were not provably real-time, or they imposed large space overheads to meet the real-time bounds. We present a mostly non-moving, dynamically defragmenting collector that overco ...

Keywords: defragmentation, read barrier, real-time scheduling, utilization

14 Fast barrier synchronization hardware

Carl J. Beckmann, Constantine D. Polychronopoulos

November 1990 **Proceedings of the 1990 ACM/IEEE conference on Supercomputing Supercomputing '90**

Publisher: IEEE Computer Society

Full text available:  [pdf\(984.65 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#)

Many recent studies have considered the importance of barrier synchronization overhead on parallel loop performance, especially for large-scale parallel machines. This paper describes a hardware scheme for supporting fast barrier synchronization. It allows barrier synchronization to be performed within a single instruction cycle for moderately sized systems, and is scalable with logarithmic increase in synchronization time. It supports a large number of concurrent barriers, and can also be used ...

15 Architectural Support for Software Transactional Memory

Bratin Saha, Ali-Reza Adl-Tabatabai, Quinn Jacobson

December 2006 **Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture MICRO 39**

Publisher: IEEE Computer Society

Full text available:  [pdf\(325.24 KB\)](#) Additional Information: [full citation](#), [abstract](#), [index terms](#)

Transactional memory provides a concurrency control mechanism that avoids many of the pitfalls of lock-based synchronization. Researchers have proposed several different implementations of transactional memory, broadly classified into software transactional memory (STM) and hardware transactional memory (HTM). Both approaches have their pros and cons: STMs provide rich and flexible transactional semantics on stock processors but incur significant overheads. HTMs, on the other hand, provide high ...

16 Optimization and real time GC: Mark-sweep or copying?: a "best of both worlds" algorithm and a hardware-supported real-time implementation

Sylvain Stanchina, Matthias Meyer

October 2007 **Proceedings of the 6th international symposium on Memory management ISMM '07**

Publisher: ACM

Full text available:  [pdf\(294.73 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Copying collectors offer a number of advantages over their mark-sweep counterparts. First, they do not have to deal with mark stacks and potential mark stack overflows. Second, they do not suffer from unpredictable fragmentation overheads since they inherently compact the heap. Third, the tospace invariant maintained by many copying collectors allows for incremental compaction and provides the basis for efficient real-time implementations. Unfortunately, however, standard copying collectors de ...

Keywords: hardware support, mark-compact collection, object-based processor architecture, real-time garbage collection

17 Optimization and real time GC: Stopless: a real-time garbage collector for multiprocessors

Filip Pizlo, Daniel Frampton, Erez Petrank, Bjarne Steensgaard
 October 2007 **Proceedings of the 6th international symposium on Memory management ISMM '07**

Publisher: ACM

Full text available:  [pdf\(303.94 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

We present Stopless: a concurrent real-time garbage collector suitable for modern multiprocessors running parallel multithreaded applications. Creating a garbage-collected environment that supports real-time on modern platforms is notoriously hard, especially if real-time implies lock-freedom. Known real-time collectors either restrict the real-time guarantees to uniprocessors only, rely on special hardware, or just give up supporting atomic operations (which are crucial for lock-free software ...

Keywords: concurrency, garbage collection, lock-free, real-time

18 Impact of Java Memory Model on Out-of-Order Multiprocessors

Tulika Mitra, Abhik Roychoudhury, Qinghua Shen

September 2004 **Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques PACT '04**

Publisher: IEEE Computer Society

Full text available:  [pdf\(168.66 KB\)](#) Additional Information: [full citation](#), [abstract](#)


The semantics of Java multithreading dictates all possible behaviors that a multithreaded Java program can exhibit on any platform. This is called the Java Memory Model (JMM) and describes the allowed reorderings among the memory operations in a thread. However, multiprocessor platforms traditionally have memory consistency models of their own. In this paper, we study the interaction between the JMM and the multiprocessor memory consistency models. In particular, memory barriers may have to be i ...

19 Implicit coscheduling: coordinated scheduling with implicit information in distributed systems

Andrea Carol Arpaci-Dusseau

August 2001 **ACM Transactions on Computer Systems (TOCS)**, Volume 19 Issue 3

Publisher: ACM Press

Full text available:  [pdf\(1.83 MB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

In modern distributed systems, coordinated time-sharing is required for communicating processes to leverage the performance of switch-based networks and low-overhead protocols. Coordinated time-sharing has traditionally been achieved with gang scheduling or explicit coscheduling, implementations of which often suffer from many deficiencies: multiple points of failure, high context-switch overheads, and poor interaction with client-server, interactive, and I/O-intensive workloads. I ...

Keywords: clusters, coscheduling, gang scheduling, networks of workstations, proportional-share scheduling, two-phase waiting

20 Faster high-level language virtual machines: Automatic feedback-directed object inlining in the java hotspot™ virtual machine

Christian Wimmer, Hanspeter Mössenböck

June 2007 **Proceedings of the 3rd international conference on Virtual execution environments VEE '07**

Publisher: ACM Press

Full text available:  [pdf\(341.49 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Object inlining is an optimization that embeds certain referenced objects into their referencing object. It reduces the costs of field accesses by eliminating unnecessary field loads. The order of objects in the heap is changed in such a way that objects that are accessed together are placed next to each other in memory so that their offset is fixed, i.e. the objects are *colocated*. This allows field loads to be replaced by address arithmetic. We implemented this optimization for ...

Keywords: cache, garbage collection, java, just-in-time compilation, object colocation, object inlining, optimization, performance

Results 1 - 20 of 200

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

The ACM Portal is published by the Association for Computing Machinery. Copyright © 2007 ACM, Inc.

[Terms of Usage](#) [Privacy Policy](#) [Code of Ethics](#) [Contact Us](#)

Useful downloads:  [Adobe Acrobat](#)  [QuickTime](#)  [Windows Media Player](#)  [Real Player](#)



USPTO

[Subscribe \(Full Service\)](#) [Register \(Limited Service, Free\)](#) [Login](#)

 Search: ☒ The ACM Digital Library ☐ The Guide

david bacon read barrier

SEARCH

THE ACM DIGITAL LIBRARY


[Feedback](#) [Report a problem](#) [Satisfaction survey](#)
Terms used: david bacon read barrier

Found 20,456 of 215,481

Sort results by

relevance

[Save results to a Binder](#)[Try an Advanced Search](#)

Display results

expanded form

[Search Tips](#)[Try this search in The ACM Guide](#)
☐ Open results in a new window

Results 1 - 20 of 200

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

Best 200 shown

Relevance scale ☐ ☐ ☐ ☐ ☐

1 [Optimization and real time GC: Stopless: a real-time garbage collector for multiprocessors](#)

Filip Pizlo, Daniel Frampton, Erez Petrank, Bjarne Steensgaard

 October 2007 **Proceedings of the 6th international symposium on Memory management ISMM '07**

Publisher: ACM

 Full text available: pdf(303.94 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

We present Stopless: a concurrent real-time garbage collector suitable for modern multiprocessors running parallel multithreaded applications. Creating a garbage-collected environment that supports real-time on modern platforms is notoriously hard, especially if real-time implies lock-freedom. Known real-time collectors either restrict the real-time guarantees to uniprocessors only, rely on special hardware, or just give up supporting atomic operations (which are crucial for lock-free software ...

Keywords: concurrency, garbage collection, lock-free, real-time

2 [A real-time garbage collector with low overhead and consistent utilization](#)

David F. Bacon, Perry Cheng, V. T. Rajan

 January 2003 **ACM SIGPLAN Notices , Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages POPL '03**, Volume 38 Issue 1

Publisher: ACM Press

 Full text available: pdf(517.37 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Now that the use of garbage collection in languages like Java is becoming widely accepted due to the safety and software engineering benefits it provides, there is significant interest in applying garbage collection to hard real-time systems. Past approaches have generally suffered from one of two major flaws: either they were not provably real-time, or they imposed large space overheads to meet the real-time bounds. We present a mostly non-moving, dynamically defragmenting collector that overco ...

Keywords: defragmentation, read barrier, real-time scheduling, utilization

3 [Compile-Time Concurrent Marking Write Barrier Removal](#)

V. Krishna Nandivada, David Detlefs

March 2005 **Proceedings of the international symposium on Code generation and optimization CGO '05**

Publisher: IEEE Computer Society

Full text available:  [pdf\(225.35 KB\)](#) Additional Information: [full citation](#), [abstract](#), [citations](#), [index terms](#)

Garbage collectors incorporating concurrent marking to cope with large live data sets and stringent pause time constraints have become common in recent years. The snapshot-at-the-beginning style of concurrent marking has several advantages over the incremental update alternative, but one main disadvantage: it requires the mutator to execute a significantly more expensive write barrier. This paper demonstrates that a large fraction of these write barriers are unnecessary, and may be eliminated by ...


4 Compiler transformations for high-performance computing



David F. Bacon, Susan L. Graham, Oliver J. Sharp

December 1994 **ACM Computing Surveys (CSUR)**, Volume 26 Issue 4

Publisher: ACM Press

Full text available:  [pdf\(6.32 MB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#), [review](#)

In the last three decades a large number of compiler transformations for optimizing programs have been implemented. Most optimizations for uniprocessors reduce the number of instructions executed by the program using transformations based on the analysis of scalar quantities and data-flow techniques. In contrast, optimizations for high-performance superscalar, vector, and parallel processors maximize parallelism and memory locality with transformations that rely on tracking the properties o ...

Keywords: compilation, dependence analysis, locality, multiprocessors, optimization, parallelism, superscalar processors, vectorization


5 Compilation Techniques for Real-Time Java Programs



Mike Fulton, Mark Stoodley

March 2007 **Proceedings of the International Symposium on Code Generation and Optimization CGO '07**

Publisher: IEEE Computer Society

Full text available:  [pdf\(275.22 KB\)](#) Additional Information: [full citation](#), [abstract](#), [index terms](#)

In this paper, we introduce the IBM® WebSphere® Real Time product, which incorporates a virtual machine that is fully Java™ compliant as well as compliant with the Real-Time Specification for Java (RTSJ). We describe IBM's real-time Java enhancements, particularly in the area of our Testarossa (TR) ahead-of-time (AOT) compiler, our TR just-in-time (JIT) compiler, and our Metronome[2] deterministic Garbage Collector (GC). The main focus of this paper is on the various techniques employed by ...

6 An on-the-fly reference-counting garbage collector for java



Yossi Levanoni, Erez Petrank

January 2006 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 28 Issue 1

Publisher: ACM Press

Full text available:  [pdf\(787.15 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Reference-counting is traditionally considered unsuitable for multiprocessor systems. According to conventional wisdom, the update of reference slots and reference-counts requires atomic or synchronized operations. In this work we demonstrate this is not the case by presenting a novel reference-counting algorithm suitable for a multiprocessor system that does not require any synchronized operation in its write barrier (not even a compare-and-swap type of synchronization). A second novelty of thi ...

Keywords: Programming languages, garbage collection, memory management, reference-counting

7 Concurrency: Write barrier elision for concurrent garbage collectors



Martin T. Vechev, David F. Bacon

October 2004 **Proceedings of the 4th international symposium on Memory management ISMM '04**

Publisher: ACM Press

Full text available: [pdf\(490.73 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Concurrent garbage collectors require write barriers to preserve consistency, but these barriers impose significant direct and indirect costs. While there has been a lot of work on optimizing write barriers, we present the first study of their elision in a concurrent collector. We show conditions under which write barriers are redundant, and describe how these conditions can be applied to both incremental update or snapshot-at-the-beginning barriers. We then evaluate the potential for write b ...

Keywords: concurrent garbage collection, write barrier

8 An on-the-fly reference counting garbage collector for Java



Yossi Levanoni, Erez Petrank

October 2001 **ACM SIGPLAN Notices , Proceedings of the 16th ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications OOPSLA '01**, Volume 36 Issue 11

Publisher: ACM Press

Full text available: [pdf\(280.30 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Reference counting is not naturally suitable for running on multiprocessors. The update of pointers and reference counts requires atomic and synchronized operations. We present a novel reference counting algorithm suitable for a multiprocessor that does not require any synchronized operation in its write barrier (not even a compare-and-swap type of synchronization). The algorithm is efficient and may complete with any tracing algorithm.

9 An efficient on-the-fly cycle collection



Harel Paz, David F. Bacon, Elliot K. Kolodner, Erez Petrank, V. T. Rajan

August 2007 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 29 Issue 4

Publisher: ACM Press

Full text available: [pdf\(952.16 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

A reference-counting garbage collector cannot reclaim unreachable cyclic structures of objects. Therefore, reference-counting collectors either use a backup tracing collector infrequently, or employ a cycle collector to reclaim cyclic structures. We propose a new *concurrent* cycle collector, one that runs concurrently with the program threads, imposing negligible pauses (of around 1ms) on a multiprocessor.


Our new collector combines a state-of-the-art cycle collector [Bacon and R ...

Keywords: Programming languages, concurrent cycle collection, garbage collection, memory management, reference counting, runtime systems

10

Languages: High-level real-time programming in Java




-  David F. Bacon, Perry Cheng, David Grove, Michael Hind, V. T. Rajan, Eran Yahav, Matthias Hauswirth, Christoph M. Kirsch, Daniel Spoonhower, Martin T. Vechev
September 2005 **Proceedings of the 5th ACM international conference on Embedded software EMSOFT '05**


Publisher: ACM Press

Full text available:  [pdf\(341.79 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Real-time systems have reached a level of complexity beyond the scaling capability of the low-level or restricted languages traditionally used for real-time programming. While Metronome garbage collection has made it practical to use Java to implement real-time systems, many challenges remain for the construction of complex real-time systems, some specific to the use of Java and others simply due to the change in scale of such systems. The goal of our current research is the creation of a comprehe ...

Keywords: WCET, allocation, scheduling, tasks, visualization

- 11 CGCEXplorer: a semi-automated search procedure for provably correct concurrent collectors 

 Martin T. Vechev, Eran Yahav, David F. Bacon, Noam Rinetzkky
June 2007 **ACM SIGPLAN Notices , Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation PLDI '07**, Volume 42 Issue 6


Publisher: ACM Press

Full text available:  [pdf\(290.80 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Concurrent garbage collectors are notoriously hard to design, implement, and verify. We present a framework for the automatic exploration of a space of concurrent mark-and-sweep collectors. In our framework, the designer specifies a set of "building blocks" from which algorithms can be constructed. These blocks reflect the designer's insights about the coordination between the collector and the mutator. Given a set of building blocks, our framework automatically explores a space of algorithms ...

Keywords: concurrent algorithms, concurrent garbage collection, synthesis, verification

- 12 Syncopation: generational real-time garbage collection in the metronome 


 David F. Bacon, Perry Cheng, David Grove, Martin T. Vechev
June 2005 **ACM SIGPLAN Notices , Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems LCTES '05**, Volume 40 Issue 7

Publisher: ACM Press

Full text available:  [pdf\(212.34 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citings](#), [index terms](#)

Real-time garbage collection has been shown to be feasible, but for programs with high allocation rates, the utilization achievable is not sufficient for some systems. Since a high allocation rate is often correlated with a more high-level, abstract programming style, the ability to provide good real-time performance for such programs will help continue to raise the level of abstraction at which real-time systems can be programmed. We have developed techniques that allow generational collection to ...

Keywords: allocation, garbage collection, real-time, scheduling

- 13 A unified theory of garbage collection 

David F. Bacon, Perry Cheng, V. T. Rajan
October 2004 **ACM SIGPLAN Notices , Proceedings of the 19th annual ACM SIGPLAN**



conference on Object-oriented programming, systems, languages, and applications OOPSLA '04, Volume 39 Issue 10

Publisher: ACM Press

Full text available: [pdf\(223.52 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Tracing and reference counting are uniformly viewed as being fundamentally different approaches to garbage collection that possess very distinct performance properties. We have implemented high-performance collectors of both types, and in the process observed that the more we optimized them, the more similarly they behaved - that they seem to share some deep structure.

We present a formulation of the two algorithms that shows that they are in fact duals of each other. Intuitively, the ...

Keywords: graph algorithms, mark-and-sweep, reference counting, tracing

14 Practical abstractions: Reflexes: abstractions for highly responsive systems



Jesper Honig Spring, Filip Pizlo, Rachid Guerraoui, Jan Vitek

June 2007 **Proceedings of the 3rd international conference on Virtual execution environments VEE '07**

Publisher: ACM Press

Full text available: [pdf\(747.33 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Commercial Java virtual machines are designed to maximize the performance of applications at the expense of predictability. High throughput garbage collection algorithms, for example, can introduce pauses of 100 milliseconds or more. We are interested in supporting applications with response times in the tens of microseconds and their integration with larger timing-oblivious applications in the same Java virtual machine. We propose Reflexes, a new abstraction for writing highly responsive sys ...

Keywords: Java virtual machine, memory management, ownership types, real-time systems

15 Virtualization and operating systems: Libra: a library operating system for a jvm in a virtualized execution environment



Glenn Ammons, Jonathan Appavoo, Maria Butrico, Dilma Da Silva, David Grove, Kiyokuni Kawachiya, Orran Krieger, Bryan Rosenburg, Eric Van Hensbergen, Robert W. Wisniewski

June 2007 **Proceedings of the 3rd international conference on Virtual execution environments VEE '07**

Publisher: ACM Press

Full text available: [pdf\(223.69 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

If the operating system could be specialized for every application, many applications would run faster. For example, Java virtual machines (JVMs) provide their own threading model and memory protection, so general-purpose operating system implementations of these abstractions are redundant. However, traditional means of transforming existing systems into specialized systems are difficult to adopt because they require replacing the entire operating system. This paper describes Libra, an execut ...

Keywords: JVM, exokernels, virtualization, xen

16 Hardware-assisted replay of multiprocessor programs



David F. Bacon, Seth Copen Goldstein

December 1991 **ACM SIGPLAN Notices , Proceedings of the 1991 ACM/ONR workshop**

on Parallel and distributed debugging PADD '91, Volume 26 Issue 12**Publisher:** ACM PressFull text available:  [pdf\(1.20 MB\)](#) Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)17 Fast interprocedural class analysis

Greg DeFouw, David Grove, Craig Chambers

January 1998 **Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages POPL '98****Publisher:** ACM PressFull text available:  [pdf\(2.03 MB\)](#) Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)18 An on-the-fly mark and sweep garbage collector based on sliding views

Hezi Azatchi, Yossi Levanoni, Harel Paz, Erez Petrank


October 2003 **ACM SIGPLAN Notices , Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications OOPSLA '03**, Volume 38 Issue 11**Publisher:** ACM PressFull text available:  [pdf\(244.12 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

With concurrent and garbage collected languages like Java and C# becoming popular, the need for a suitable non-intrusive, efficient, and concurrent multiprocessor garbage collector has become acute. We propose a novel mark and sweep on-the-fly algorithm based on the sliding views mechanism of Levanoni and Petrank. We have implemented our collector on the Jikes Java Virtual Machine running on a Netfinity multiprocessor and compared it to the concurrent algorithm and to the stop-the-world collecto ...

Keywords: concurrent garbage collection, garbage collection, memory management, on-the-fly garbage collection, runtime systems

19 New garbage collection algorithms and strategies: Garbage-first garbage collection

David Detlefs, Christine Flood, Steve Heller, Tony Printezis

October 2004 **Proceedings of the 4th international symposium on Memory management ISMM '04****Publisher:** ACM PressFull text available:  [pdf\(199.59 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

<i>Garbage-First</i> is a server-style garbage collector, targeted for multi-processors with large memories, that meets a soft real-time goal with high probability, while achieving high throughput. Whole-heap operations, such as global marking, are performed concurrently with mutation, to prevent interruptions proportional to heap or live-data size. Concurrent marking both provides collection "completeness" and identifies regions ripe for reclamation via compacting evacuation. This ev ...

Keywords: concurrent garbrage collection, garbage collection, garbage-first garbage collection, parallel garbage collection, soft real-time garbage collection

20

Implementations: Design and implementation of a comprehensive real-time java virtual machine

Joshua Auerbach, David F. Bacon, Bob Blainey, Perry Cheng, Michael Dawson, Mike Fulton,

David Grove, Darren Hart, Mark Stoodley

September 2007 **Proceedings of the 7th ACM & IEEE international conference on Embedded software EMSOFT '07**

Publisher: ACM Press

Full text available:  [pdf\(405.84 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

The emergence of standards for programming real-time systems in Java has encouraged many developers to consider its use for systems previously only built using C, Ada, or assembly language. However, the RTSJ standard in isolation leaves many important problems unaddressed, and suffers from some serious problems in usability and safety.

As a result, the use of Java for real-time programming has continued to be viewed as risky and adoption has been slow.

In this paper we provide a ...

Keywords: AOT, JIT, JVM, garbage collection, java, real time

Results 1 - 20 of 200

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

The ACM Portal is published by the Association for Computing Machinery. Copyright © 2007 ACM, Inc.

[Terms of Usage](#) [Privacy Policy](#) [Code of Ethics](#) [Contact Us](#)

Useful downloads:  [Adobe Acrobat](#)  [QuickTime](#)  [Windows Media Player](#)  [Real Player](#)



USPTO

[Subscribe \(Full Service\)](#) [Register \(Limited Service, Free\)](#) [Login](#)

Search: ☒ The ACM Digital Library ☐ The Guide

read barrier null reference check



[Feedback](#) [Report a problem](#) [Satisfaction survey](#)

Terms used: read barrier null reference check

Found 109,100 of 215,481

Sort results by

relevance



[Save results to a Binder](#)

[Try an Advanced Search](#)

Display results

expanded form



[Search Tips](#)

[Try this search in The ACM Guide](#)

☐ Open results in a new window

Results 1 - 20 of 200

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

Best 200 shown

Relevance scale ☐ ☐ ☐ ☐ ☐

1 [An on-the-fly reference-counting garbage collector for java](#)



Yossi Levanoni, Erez Petrank

January 2006 **ACM Transactions on Programming Languages and Systems (TOPLAS)**,

Volume 28 Issue 1

Publisher: ACM Press

Full text available: [pdf\(787.15 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Reference-counting is traditionally considered unsuitable for multiprocessor systems. According to conventional wisdom, the update of reference slots and reference-counts requires atomic or synchronized operations. In this work we demonstrate this is not the case by presenting a novel reference-counting algorithm suitable for a multiprocessor system that does not require any synchronized operation in its write barrier (not even a compare-and-swap type of synchronization). A second novelty of thi ...

Keywords: Programming languages, garbage collection, memory management, reference-counting

2 [Effective null pointer check elimination utilizing hardware trap](#)



Motohiro Kawahito, Hideaki Komatsu, Toshio Nakatani

November 2000 **ACM SIGOPS Operating Systems Review , ACM SIGARCH Computer Architecture News , Proceedings of the ninth international conference on Architectural support for programming languages and operating systems ASPLOS-IX**, Volume 34 , 28 Issue 5 , 5

Publisher: ACM

Full text available: [pdf\(307.23 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [cited by](#), [index terms](#)

We present a new algorithm for eliminating null pointer checks from programs written in Java™. Our new algorithm is split into two phases. In the first phase, it moves null checks backward, and it is iterated for a few times with other optimizations to eliminate redundant null checks and maximize the effectiveness of other optimizations. In the second phase, it moves null checks forward and converts many null checks to hardware traps in order to minimize the execution cost of the remai ...

3 [Effective null pointer check elimination utilizing hardware trap](#)



Motohiro Kawahito, Hideaki Komatsu, Toshio Nakatani

November 2000 **ACM SIGPLAN Notices**, Volume 35 Issue 11

Publisher: ACM Press

Full text available:  [pdf\(307.23 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

We present a new algorithm for eliminating null pointer checks from programs written in Java[®]. Our new algorithm is split into two phases. In the first phase, it moves null checks backward, and it is iterated for a few times with other optimizations to eliminate redundant null checks and maximize the effectiveness of other optimizations. In the second phase, it moves null checks forward and converts many null checks to hardware traps in order to minimize the execution cost of the remain ...

4 Garbage collection: A true hardware read barrier

Matthias Meyer

June 2006 **Proceedings of the 5th international symposium on Memory management ISMM '06**

Publisher: ACM

Full text available:  [pdf\(991.66 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Read barriers synchronize compacting garbage collection and application processing in a simple yet elegant way. Unfortunately, read barrier checks are expensive to implement in software, and even with hardware support, the clustering of read barrier faults irregularly impairs application progress to an unacceptable extent. For this reason, read barriers are often considered unsuitable for hard real-time systems. In this paper, we introduce a novel hardware read barrier design for an object-based ...

Keywords: hardware support, object-based processor architecture, read barrier, real-time garbage collection

5 Compile-Time Concurrent Marking Write Barrier Removal

V. Krishna Nandivada, David Detlefs

March 2005 **Proceedings of the international symposium on Code generation and optimization CGO '05**

Publisher: IEEE Computer Society

Full text available:  [pdf\(225.35 KB\)](#) Additional Information: [full citation](#), [abstract](#), [citations](#), [index terms](#)

Garbage collectors incorporating concurrent marking to cope with large live data sets and stringent pause time constraints have become common in recent years. The snapshot-at-the-beginning style of concurrent marking has several advantages over the incremental update alternative, but one main disadvantage: it requires the mutator to execute a significantly more expensive write barrier. This paper demonstrates that a large fraction of these write barriers are unnecessary, and may be eliminated by ...


6 Introducing Ada 9X



John Barnes

November 1993 **ACM SIGAda Ada Letters**, Volume XIII Issue 6

Publisher: ACM Press

Full text available:  [pdf\(4.39 MB\)](#) Additional Information: [full citation](#), [citations](#), [index terms](#)

7 Concurrent garbage collection using hardware-assisted profiling



Timothy H. Heil, James E. Smith

October 2000 **ACM SIGPLAN Notices , Proceedings of the 2nd international symposium on Memory management ISMM '00**, Volume 36 Issue 1

Publisher: ACM Press

Full text available:  [pdf\(1.74 MB\)](#) Additional Information: [full citation](#), [abstract](#), [citations](#), [index terms](#)

In the presence of on-chip multithreading, a Virtual Machine (VM) implementation can


readily take advantage of *service threads* for enhancing performance by performing tasks such as profile collection and analysis, dynamic optimization, and garbage collection concurrently with program execution. In this context, a hardware-assisted profiling mechanism is proposed. The *Relational Profiling Architecture* (RPA) is designed from the top down. RPA is based on a relational model similar ...

8 Compilation Techniques for Real-Time Java Programs

Mike Fulton, Mark Stoodley

March 2007 **Proceedings of the International Symposium on Code Generation and Optimization CGO '07**

Publisher: IEEE Computer Society

Full text available:  [pdf\(275.22 KB\)](#) Additional Information: [full citation](#), [abstract](#), [index terms](#)

In this paper, we introduce the IBM® WebSphere® Real Time product, which incorporates a virtual machine that is fully Java™ compliant as well as compliant with the Real-Time Specification for Java (RTSJ). We describe IBM's real-time Java enhancements, particularly in the area of our Testarossa (TR) ahead-of-time (AOT) compiler, our TR just-in-time (JIT) compiler, and our Metronome[2] deterministic Garbage Collector (GC). The main focus of this paper is on the various techniques employed by ...

9 CGCExplorer: a semi-automated search procedure for provably correct concurrent collectors

Martin T. Vechev, Eran Yahav, David F. Bacon, Noam Rinetzky

June 2007 **ACM SIGPLAN Notices , Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation PLDI '07**, Volume 42 Issue 6

Publisher: ACM Press

Full text available:  [pdf\(290.80 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Concurrent garbage collectors are notoriously hard to design, implement, and verify. We present a framework for the automatic exploration of a space of concurrent mark-and-sweep collectors. In our framework, the designer specifies a set of "building blocks" from which algorithms can be constructed. These blocks reflect the designer's insights about the coordination between the collector and the mutator. Given a set of building blocks, our framework automatically explores a space of algorithms ...

Keywords: concurrent algorithms, concurrent garbage collection, synthesis, verification

10 DITTO: automatic incrementalization of data structure invariant checks (in Java)

Ajeet Shankar, Rastislav Bodík

June 2007 **ACM SIGPLAN Notices , Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation PLDI '07**, Volume 42 Issue 6

Publisher: ACM Press

Full text available:  [pdf\(258.66 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

We present DITTO, an automatic incrementalizer for dynamic, side-effect-free data structure invariant checks. Incrementalization speeds up the execution of a check by reusing its previous executions, checking the invariant anew only the changed parts of the data structure. DITTO exploits properties specific to the domain of invariant checks to automate and simplify the process without restricting what mutations the program can perform. Our incrementalizer works for modern imperative languages ...

Keywords: automatic, data structure invariants, dynamic optimization, incrementalization, optimistic memoization, program analysis

11 Anatomy of LISP

John Allen
January 1978 Book

Publisher: McGraw-Hill, Inc.

Additional Information: [full citation](#), [abstract](#), [references](#), [cited by](#), [index terms](#)

This text is nominally about LISP and data structures. However, in the process it covers much broader areas of computer science. The author has long felt that the beginning student of computer science has been getting 'a distorted and disjointed picture of the field. In some ways this confusion is natural; the field has been growing at such a rapid rate that few are prepared to be judged experts in all areas of the discipline. The current alternative seems to be to give a few introductory cou ...

12 The multics system: an examination of its structure

Elliott I. Organick
January 1972 Book

Publisher: MIT Press

Additional Information: [full citation](#), [abstract](#), [references](#), [cited by](#), [index terms](#)

This volume provides an overview of the Multics system developed at M.I.T.--a time-shared, general purpose utility like system with third-generation software. The advantage that this new system has over its predecessors lies in its expanded capacity to manipulate and file information on several levels and to police and control access to data in its various files. On the invitation of M.I.T.'s Project MAC, Elliott Organick developed over a period of years an explanation of the workings, concep ...

13 An on-the-fly reference counting garbage collector for Java

Yossi Levanoni, Erez Petrank

October 2001 **ACM SIGPLAN Notices , Proceedings of the 16th ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications OOPSLA '01**, Volume 36 Issue 11

Publisher: ACM Press

Full text available: [pdf\(280.30 KB\)](#)

Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Reference counting is not naturally suitable for running on multiprocessors. The update of pointers and reference counts requires atomic and synchronized operations. We present a novel reference counting algorithm suitable for a multiprocessor that does not require any synchronized operation in its write barrier (not even a compare-and-swap type of synchronization). The algorithm is efficient and may complete with any tracing algorithm.

14 An on-the-fly mark and sweep garbage collector based on sliding views

Hezi Azatchi, Yossi Levanoni, Harel Paz, Erez Petrank

October 2003 **ACM SIGPLAN Notices , Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications OOPSLA '03**, Volume 38 Issue 11

Publisher: ACM Press

Full text available: [pdf\(244.12 KB\)](#)

Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

With concurrent and garbage collected languages like Java and C# becoming popular, the need for a suitable non-intrusive, efficient, and concurrent multiprocessor garbage collector has become acute. We propose a novel mark and sweep on-the-fly algorithm based on the sliding views mechanism of Levanoni and Petrank. We have implemented our collector on the Jikes Java Virtual Machine running on a Netfinity multiprocessor and compared it to the concurrent algorithm and to the stop-the-world collecto ...

Keywords: concurrent garbage collection, garbage collection, memory management, on-the-fly garbage collection, runtime systems

15 Guide for the use of the Ada Ravenscar Profile in high integrity systems



Alan Burns, Brian Dobbing, Tullio Vardanega

June 2004 **ACM SIGAda Ada Letters**, Volume XXIV Issue 2

Publisher: ACM Press

Full text available: pdf(548.17 KB) Additional Information: [full citation](#), [references](#)



16 Adaptive techniques: Optimistic stack allocation for java-like languages

Erik Corry

June 2006 **Proceedings of the 5th international symposium on Memory management ISMM '06**

Publisher: ACM

Full text available: pdf(155.23 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Stack allocation of objects offers more efficient use of cache memories on modern computers, but finding objects that can be safely stack allocated is difficult, as interprocedural escape analysis is imprecise in the presence of virtual method dispatch and dynamic class loading. We present a new technique for doing optimistic stack allocation of objects. Our technique does not require interprocedural analysis and is effective in the presence of dynamic class loading, reflection and exception han ...

Keywords: Java, garbage collection, stack allocation



17 Compiler construction: an advanced course

F. L. Bauer, F. L. De Remer, M. Griffiths, U. Hill, J. J. Horning, C. H. A. Koster, W. M. McKeeman, P. C. Poole, W. M. Waite, G. Goos, J. Hartmanis
January 1974 Book

Publisher: Springer-Verlag New York, Inc.

Additional Information: [full citation](#), [abstract](#), [references](#), [cited by](#)

The Advanced Course took place from March 4 to 15, 1974 and was organized by the Mathematical Institute of the Technical University of Munich and the Leibniz Computing Center of the Bavarian Academy of Sciences, in co-operation with the European Communities, sponsored by the Ministry for Research and Technology of the Federal Republic of Germany and by the European Research Office, London.



18 Implementation techniques: Barriers: friend or foe?



Stephen M. Blackburn, Antony L. Hosking

October 2004 **Proceedings of the 4th international symposium on Memory management ISMM '04**

Publisher: ACM Press

Full text available: pdf(137.10 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)


Modern garbage collectors rely on read and write barriers imposed on heap accesses by the mutator, to keep track of references between different regions of the garbage collected heap, and to synchronize actions of the mutator with those of the collector. It has been a long-standing untested assumption that barriers impose significant overhead to garbage-collected applications. As a result, researchers have devoted effort to



development of optimization approaches for elimination of unnecessary ...


Keywords: garbage collection, java, memory management, write barriers

19 Nonblocking memory management support for dynamic-sized data structures

 Maurice Herlihy, Victor Luchangco, Paul Martin, Mark Moir

May 2005 **ACM Transactions on Computer Systems (TOCS)**, Volume 23 Issue 2

Publisher: ACM Press

Full text available:  [pdf\(944.89 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#), [review](#)

Conventional dynamic memory management methods interact poorly with lock-free synchronization. In this article, we introduce novel techniques that allow lock-free data structures to allocate and free memory dynamically using any thread-safe memory management library. Our mechanisms are lock-free in the sense that they do not allow a thread to be prevented from allocating or freeing memory by the failure or delay of other threads. We demonstrate the utility of these techniques by showing how to m ...

Keywords: Multiprocessors, concurrent data structures, dynamic data structures, memory management, nonblocking synchronization

20 Session 1: Safe memory reclamation for dynamic lock-free objects using atomic reads and writes

 Maged M. Michael

July 2002 **Proceedings of the twenty-first annual symposium on Principles of distributed computing PODC '02**

Publisher: ACM Press

Full text available:  [pdf\(1.13 MB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#)

A major obstacle to the wide use of lock-free data structures, despite their many performance and reliability advantages, is the absence of a practical lock-free method for reclaiming the memory of dynamic nodes removed from dynamic lock-free objects for arbitrary reuse. The only prior lock-free memory reclamation method depends on the DCAS atomic primitive, which is not supported on any current processor architecture. Other memory management methods are blocking, require special operating system ...

Results 1 - 20 of 200

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

The ACM Portal is published by the Association for Computing Machinery. Copyright © 2007 ACM, Inc.

[Terms of Usage](#) [Privacy Policy](#) [Code of Ethics](#) [Contact Us](#)

Useful downloads:  [Adobe Acrobat](#)  [QuickTime](#)  [Windows Media Player](#)  [Real Player](#)